Text handling on a general purpose computer

M. D. Martin

There are two ways of using computers in print. The first is to take a small computer, place it firmly, if metaphorically, inside the conventional composing room set-up, and employ it as a special purpose device for the benefit of the printer alone. The second is to take the composing machine—metaphorically or literally—out of the print shop and to treat it as an extra peripheral to a general purpose computer. In the one case, the computer becomes a new and powerful printing tool; in the other, the composition process becomes just part of an overall data processing pattern.

In this paper I want to outline, in a theoretical way, the architecture of a text handling system, give some practical reasons for some of its features, and say something about how we are implementing it. Some parts of the system are still under active development. One of our main objectives has been to produce an open-ended system. We believe it can be extended in many directions without altering its basic assumptions and its basic structures.

We began by rejecting the idea of a computer typesetting 'package'. The range of composing equipment on the one hand and the range of printing applications on the other are too wide for such an approach. We are developing a series of modular programs and library subroutines interlinked by a common code structure for the representation of printable text within the computer. Apart from those routines which are purely concerned with system housekeeping, each module is related either to a particular composing function, or to a particular composing machine, or to both.

The computer typesetting process can be broken down into a few fundamental operations. These are *input, storage, display* (under which heading I include hard copy proofs and CRT systems), *editing and correction, line composing* ('justification'), *page makeup* and *output* in composing machine format. They do not necessarily occur in that order. It is clear that, of these operations, the first three—input, storage and display—are completely independent of the composing equipment. Output is entirely dependent on composing machines. If we separate the insertion of corrections from the re-formatting process which they usually demand, then editing also becomes an independent function. To go further, we can regard even line and page composing as logical processes, quite independent of typesetting hardware, provided there is enough information about the machine and type faces in use.

This point of view offers two advantages. One is that we do not have to re-program basic composing functions in order to operate with a different machine. Additionally, the composition routines are generally simpler, much of their work having been loaded on to the output program. This has the incidental merit, in a multi-pass system, that there is more processing to be done on the final, paper tape limited run and less on intermediate runs which will most often be processor limited. Generalised composing routines need information about the machine and fonts to be used. We have written programs which generate this information in a standard form for all composing machines having not more than four duplexed alphabets on-line at one time—eight alphabets in all. By 'alphabet' I mean the ninety or so characters of the printer's alphabet. We also generate output translation tables in which the format varies for the different devices. The tables are stored together in a single library on backing storage and are called out by a job specification. In a multi-pass system the type tables in use are written at the head of the text file so that there is no need to access the library on any subsequent run unless the job specification is changed.

For the rest of this paper I shall look at the internal structure of the computer text handling system.

Common code structure

The code structure which we are using to handle text within 1900 Series computers is strictly internal, and imposes almost no special requirements on keyboarding methods. An initial translation module allows any six or seven-channel paper tape code to be converted to internal code.

Our common code has two forms. The 'primary' form occurs at input when text and format information is represented as a simple linear string. The 'intermediate' form is generated when text has been structured for storage on tape or discs. In either case common code recognises three distinct kinds of information: printable text, format control messages and non-printable data. These are represented in the primary code by supershift characters. We refer to them as 'print shift' 'format' shift and 'non-print' shift. 'Print' and 'format' shifts speak for themselves. 'Non-print' shift allows the user to insert information which is required for other data processing purposes, but which may be disregarded when text is output to the composing machine. We have the option of eliminating non-print information when the text file is first presented for typesetting, or, with some restrictions, we can retain it through all intermediate stages of composition. In this case it will be skipped automatically at output. The user alone decides what, if anything, he wants to insert as 'non-printable information'. The advantage of building this facility in from the

very start is that basic typesetting programs will not require alteration if they are being used in association with other data processing operations.

For example, consider a newspaper system where the computer may be used for composing and invoicing classified ads. It will be possible to attach an account number, appearance dates and other relevant information to each advertisement. This is then accessible for accounting routines, and can be updated whenever the ad is extracted for printing; but it is disregarded by the standard program module which translates the stored records into composing machine format. Similarly, we can store non-printable keys for sorting or information retrieval, and we have in mind automatic indexing techniques based on associating a single non-print character with those words or phrases in the text which are to appear in the computer-generated index.

Both format and non-print shift codes are partly dependent on the character immediately following the shift. If this character is a letter of the alphabet, the system remains in format or non-print shift until a further shift code is read. On the other hand, any non-alphabetic character causes an immediate and automatic return to print shift. This technique allows an economic representation of elementary function codes, such as changes of alphabet, quad left etc, and of single non-print codes. At the same time an open-ended set of longer format control messages can be devised. These may be introduced by a suitable mnemonic.

One might criticise the supershift system on the grounds of increased storage requirement when, effectively, all format control codes use 12 bits against the 6 bits used for all printable characters. In fact the ratio of format codes to printable symbols is such that this increase is seldom very significant and we can set against this several positive advantages. One is that it is very easy to separate format codes from text, eg to generate an alphabetic sorting key from printable text. Another is that if we are scanning text and looking for format control codes, as the programs often are, a single test suffices to determine that a given symbol is *not* a format shift code. This tends to improve processing time. Also by separating print and format information in this way we make available a much larger set of codes for each class of data.

So much for the 'primary' form of our internal code structure. Before I discuss the storage of text in its 'intermediate' format, I want to mention paper tape input arrangements. I have already said that the internal code structure need not be exactly reflected in the keyboarding method used to prepare text for typesetting. We have felt it advisable not to limit ourselves to any particular keyboard or paper tape code. Accordingly, the paper tape input module includes a table-driven translation routine which can be amended to read any six or seven-channel code. In one version of the input routine it is possible to define individual keys to represent strings of characters. This may be done on a permanent basis (for example to store job specifications which are in everyday use), or, if the keyboard operator knows that he has a spare key, he can define it for the duration of the current job, or redefine it as often as he wishes during the job to save keystrokes on repetitive data.

I have described the 'primary' common code format, where input text is represented as a continuous string. Clearly when information is stored on magnetic tape or discs it must be structured in a more complex manner. We find it convenient to use the line as fundamental record for all intermediate storage. This applies even when the text is still unprocessed, although in this case the division into 'lines' is somewhat arbitrary. The important thing is that the record format is identical for non-print information, unprocessed text, processed text and long format control messages.

The line consists of a fixed-length header and a variable-length string of text, which is stored in a form identical to that used for primary text. The difference is that, at this stage, long non-print messages and long format control messages have been separated from the text and are stored as lines in their own right. Some additional format lines may have been spontaneously generated by the composition routines, for example end-of-page markers. Thus all major format changes are represented by unique records in the text file. If the composing equipment is such that the different type styles or sizes required in one job cannot be encompassed in a single run, it is very easy to sort the file by reference to these format lines and divide the job into sections corresponding to runs on the composing machine. On the other hand, minor format changes, such as a change of alphabet within the magazine, or change of matrix case, are represented by single control codes held within the text.

The header preceding a line of processed text is used to store various parameters associated with that line. These include the spacing requirements for justified lines, which are expressed simply in terms of the number of variable spaces and the number of units of space to be taken up. Thus they remain independent of the composing equipment until the final output stage is reached. The header structure also permits an easy representation of multi-column formats. We can indicate that several records are to be composed sequentially across a single line by setting a count which in the leftmost record is equal to one less than the number of columns, and decreases to zero in the last record of that line. Single-column setting is a special subset of this system in which all records hold a zero count.

As soon as we have arranged the processed or unprocessed text into an intermediate common code structure, we can display it on the line printer by using a single standard proof print subroutine. Theoretically common code allows us a set of more than a hundred printable characters per alphabet—more than enough to cover the usual printer's alphabet, and enough to give us a certain flexibility in allocating codes for non-standard symbols. The line printer has a character set of only 64. We have adopted the by no means original technique of printing two lines per line of text and indicating format changes by codes placed in the upper line; but we have further extended this convention so that no format information whatsoever appears in the lower line—that is in the text line.

We also follow a rule that all characters which have an exact representation on the printer are located in lower case or unshift. All additional symbols are located in upper case or shift. Shift is

identified by a dot above the character in question. The reader then knows that any non-alphabetic character flagged with a dot represents a special symbol not available on the line printer, the only exceptions being a few codes like em space and period which appear in both shift and unshift as they do in many composing machine layouts. This proof print is the basis for the correction process. The technique is one of merging correction messages with an intermediate text file to produce a character stream which, as far as all formatting routines are concerned, looks the same as the primary stream of unstructured text. The same correction subroutine will handle justified or unjustified text. The same composition programs will handle a text stream coming straight from an input device, or one from the correction subroutine.

These programs are all written in machine assembly language. Most of them take the form of library subroutines which, like any standard set of commercial and mathematical subroutines, are accessible at assembly language level. A typesetting program may then consist of a simple series of subroutine calls, or there is ample scope for adding special coding for particular applications. I have deliberately not elaborated on the mechanics of justification, hyphenation, correction and so on, although, in practice, a very high proportion of our programming has to be concerned with these. There is not very much to be said here that has not been said before. What is perhaps new and interesting is the overall systems approach: first that we are producing a typesetting system which is built from the ground up to handle, and, where necessary, to reject non-printable information, and this will enable us to use standard composition routines for non-standard applications. (In print there is no such thing as a standard application.) Second, that we are building it in a generalised and entirely modular way, to give both ourselves and the user the greatest possible flexibility.